1. **소스코드 및 주석**
   1. **1번 문제**

#include <iostream>

#define int\_max 1000000000

using namespace std;

// 최적의 괄호 묶기 순서를 출력하는 함수

void print\_optimal\_parens(int\*\* S, int i, int j) {

if (i == j) {

cout << "A" << i;

}

else {

cout << "(";

print\_optimal\_parens(S, i, S[i][j]); // 왼쪽 분할

cout << " \* ";

print\_optimal\_parens(S, S[i][j] + 1, j); // 오른쪽 분할

cout << ")";

}

}

int main(void)

{

int n = 4;

// cin >> N;

int\* d = new int[n + 1];

int\*\* C = new int\* [n + 1];

int\*\* S = new int\* [n + 1];

for (int i = 0; i < n + 1; i++) {

C[i] = new int[n + 1];

S[i] = new int[n + 1];

for (int j = 0; j < n + 1; j++) {

if (i == j)

C[i][j] = 0;

else

C[i][j] = int\_max;

S[i][j] = 0;

}

}

int\* mul\_sum = new int[n];

int final\_solution;

int temp = 0;

d[0] = 10;

d[1] = 20;

d[2] = 5;

d[3] = 15;

d[4] = 30;

for (int L = 1; L <= n - 1; L++) {

for (int i = 1; i <= n - L; i++) {

int j = i + L;

for (int k = i; k < j; k++) {

temp = C[i][k] + C[k + 1][j] + d[i - 1] \* d[k] \* d[j];

if (temp < C[i][j]) {

C[i][j] = temp;

S[i][j] = k;

}

}

}

}

cout << "Chained Matrix Multiplication\n===================================\n";

cout << "M\t1\t2\t3\t4\n";

for (int i = 1; i <= n; i++) {

cout << i << "\t";

for (int j = 1; j <= n; j++) {

if (C[i][j] == int\_max) cout << "0\t";

else cout << C[i][j] << "\t";

}

cout << endl;

}

final\_solution = C[1][n];

cout << "Final Solution : " << final\_solution << endl << endl;

cout << "Implicit Order for Matrix Multiplication : ";

print\_optimal\_parens(S, 1, n);

cout << endl;

for (int i = 0; i <= n; i++) {

delete[] C[i];

delete[] S[i];

}

delete[] C;

delete[] S;

delete[] d;

return 0;

}

**1-2. 2번 문제**

#include <iostream>

#include <cstdlib>

#include <ctime>

#include <vector>

#include <fstream>

#define int\_max 1000000000

using namespace std;

// 행렬 곱셈 최소 연산 횟수 구하기

int matrix\_chain\_order(int\* d, int n) {

int\*\* C = new int\* [n + 1];

int\*\* S = new int\* [n + 1];

for (int i = 0; i <= n; i++) {

C[i] = new int[n + 1];

S[i] = new int[n + 1];

for (int j = 0; j <= n; j++) {

if (i == j)

C[i][j] = 0;

else

C[i][j] = int\_max;

S[i][j] = 0;

}

}

int temp = 0;

for (int L = 1; L <= n - 1; L++) {

for (int i = 1; i <= n - L; i++) {

int j = i + L;

for (int k = i; k < j; k++) {

temp = C[i][k] + C[k + 1][j] + d[i - 1] \* d[k] \* d[j];

if (temp < C[i][j]) {

C[i][j] = temp;

S[i][j] = k;

}

}

}

}

int result = C[1][n];

for (int i = 0; i <= n; i++) {

delete[] C[i];

delete[] S[i];

}

delete[] C;

delete[] S;

return result;

}

int main(void) {

srand(time(0));

int n\_array[6] = { 5, 6, 7, 8, 9, 10 };

vector<int> operations;

for (int n : n\_array) {

int\* d = new int[n + 1];

for (int i = 0; i <= n; i++) {

d[i] = rand() % 20 + 1;

}

int min\_operations = matrix\_chain\_order(d, n);

operations.push\_back(min\_operations);

delete[] d;

}

cout << "Matrix size vs. Operations\n==========================\n";

cout << "Matrix Count\tOperations\n";

for (auto i = 0; i < 6; i++) {

cout << n\_array[i] << "\t\t" << operations[i] << endl;

}

// 결과를 CSV 파일로 저장

ofstream file("matrix\_operations.csv");

file << "Matrix Count,Operations\n";

for (auto i = 0; i < 6; i++) {

file << n\_array[i] << "," << operations[i] << endl;

}

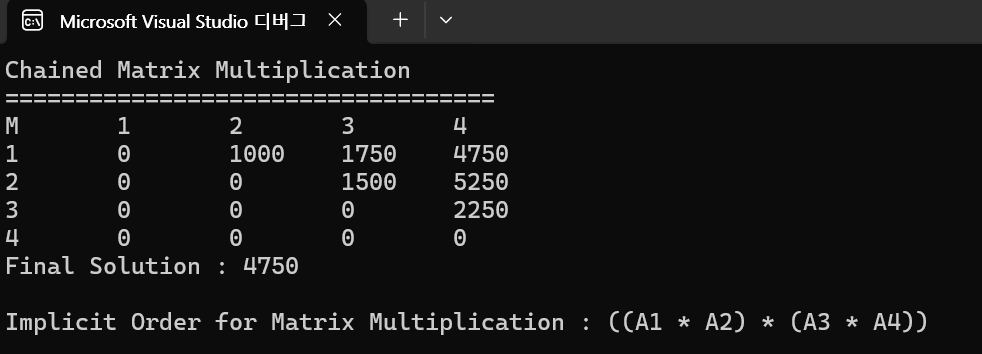
file.close();

return 0;

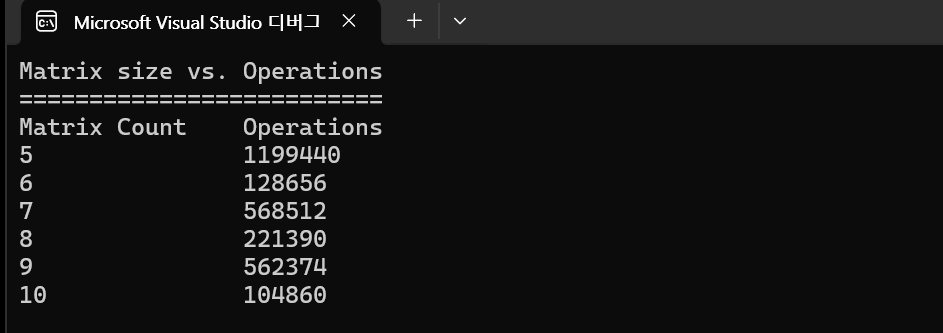
}

1. **실행화면 캡처**

**2-1. 1번 문제**

****

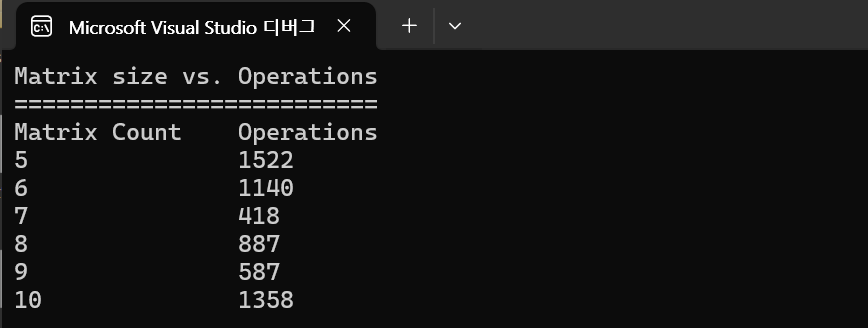
**2-2. 2번 문제**

****

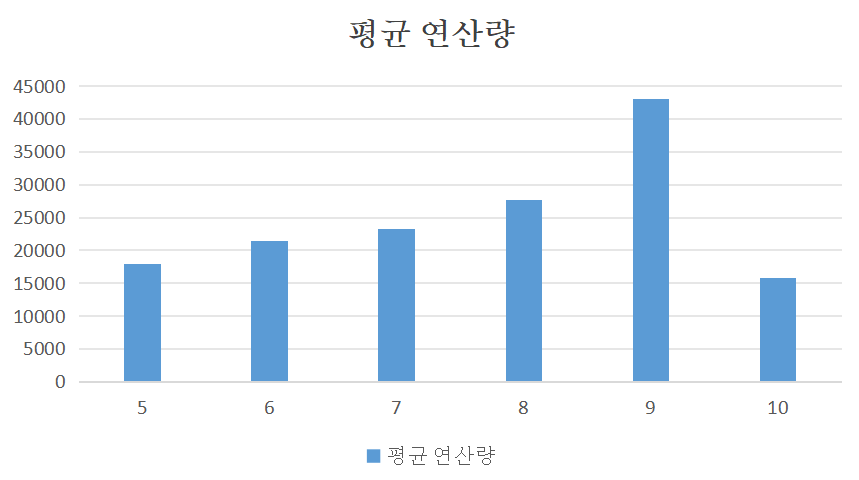
1. **d[i]이 1~100의 랜덤값일때, 매트릭스 개수가 증가하여도 연산량이 감소하는 모양**

**2. 10번 출력한 후 그래프로 분석하여도 패턴이 보이지않음.**

1. **너무 큰범위의 랜덤값이라 패턴을 찾을 수 없다고 판단하였음**

****

1. **1~20사이의 랜덤값으로 조정한 후 출력**
2. **이번에도, 10번 출력한 후 평균값을 그래프로 그려보았음**

****

1. **시간 복잡도가 n^3이므로 기하급수적으로 증가할거란 예측과 다르게 10에서 갑자기 감소하는 경향을 보였음**
2. **1~20개의 랜덤값일때, 매트릭스가 10개가 되면 더 효율적인 연산 방법을 찾아서 연산량이 오히려 감소한다고 판단**

**3.고찰**

* **매트릭스 개수가 증가하면 기하급수적으로 연산량이 증가할 거라는 예상과 다른 패턴을 확인하였음**
* **이미, 계산을 한 테이블에서 동적으로 최적의 계산을 찾는 방법이므로, 패턴을 찾기 매우 어렵다고 생각함**